A PBL Project Report On

Tic Tac Toe using Machine Learning

Submitted to JNTU HYDERABAD
In Partial Fulfilment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY IN
COMPUTER SCIENCE AND ENGINEERING

Submitted By

B. Bala Bhaskar-208R1A05C7

P.R. Dhanush Reddy-208R1A05G8

CH. Nithish Kumar– 208R1A05D3

T. Rohith Reddy – 208R1A05H5

Under the Esteemed guidance of
Mrs. G.Sumalatha

Assistant Professor, Department of CSE

Department of Computer Science & Engineering

CMR ENGINEERING COLLEGE

(Approved by AICTE, NEW DELHI, Affiliated to JNTU, Hyderabad)

Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401

2022-2023

CMR ENGINEERING COLLEGE

(Accredited by NBA, Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad) Kandlakoya, Medchal Road, Hyderabad-501 401)

Department of Computer Science & Engineering

<u>CERTIFICATE</u>

This is to certify that the PBL entitled **"Tic Tac Toe using Machine Learning"** is a Bonafede work carried out by

B. Bala Bhaskar – 208R1A05C7
P.R. Dhanush Reddy – 208R1A05G8
CH. Nithish Kumar – 208R1A05D3
T.Rohith Reddy – 208R1A05H5

In partial fulfilment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** from, CMR Engineering College, affiliated to JNTU, Hyderabad, under our guidance and supervision. The results presented in this seminar have been verified and are found to be satisfactory. The results embodied in this seminar have not been submitted to any other university for the award of any other degree or diploma

Internal Guide	Head of the Department
Mrs. G. Sumalatha	**Dr. Sheo Kumar**
Assistant Professor	Professor & H.O.D
Department of CSE CMREC,	Department of CSE CMREC,
Hyderabad.	Hyderabad

DECLARATION

This is to certify that the work reported in the present project entitled "Tic Tac Toe using Machine Learning" is a record of Bonafede work done by me in the Department of Computer Science and Engineering (AI & ML), CMR Engineering College. The reports are based on the project work done entirely by me and not copied from any other source. I submit my project for further development by any interested students who share similar interests to improve the project in the future. The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

B.Bala Bhaskar-208R1A05C7
P.R.Dhanush Reddy-208R1A05G8
CH.Nithish kumar-208R1A05D3
T.Rohith Reddy-208R1A05H5

1.INTRODUCTION

Tic Tac Toe is a simple yet challenging game that has been played for centuries. The game involves two players, X and O, who take turns marking spaces in a 3x3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. While the game's rules are straightforward, playing optimally is a challenging task that requires strategic thinking and foresight

In recent years, machine learning (ML) algorithms have shown great promise in solving complex problems and achieving optimal outcomes in various applications. In this project, we aim to develop a computer player that can play the Tic Tac Toe game optimally using ML algorithms. The computer player should be able to make intelligent moves based on the current state of the game and predict the opponent's moves to ensure victory or a draw.

The project involves several key steps, such as data collection and preprocessing, model selection and training, integration with the game interface, and testing and evaluation. By implementing these steps, we hope to gain a deeper understanding of various ML techniques and their applications in solving real-world problems. The project also has the potential to be extended to other games and applications, such as chess, checkers, or even self-driving cars.

2. PROBLEM STATEMENT

Our goal is to develop a computer player that can play Tic Tac Toe optimally using ML algorithms. The computer player should be able to make intelligent moves based on the current state of the game and predict the opponent's moves to ensure victory or a draw. By doing so, we hope to provide a reliable and efficient method for playing Tic Tac Toe that can be extended to other games and applications in the future.The problem we aim to address in this project is the lack of a reliable and efficient method for playing Tic Tac Toe optimally. While some existing computer players can play the game, they often rely on brute-force algorithms that search all possible moves and evaluate their outcomes, which can be computationally expensive and time-consuming

3.

EXISTING SYSTEM

RULE-BASED SYSTEMS:

These are deterministic algorithms that follow predefined rules to make moves in Tic Tac Toe. For example, a rule-based system may prfioritize placing a mark in a winning position, blocking the opponent's winning move, or choosing a random move from the available options. Rule-based systems are typically simple to implement but may not be very adaptive or optimal in all game situationsQlearning is a model-free reinforcement learning algorithm which is generally used to learn the best action for an agent to take given a particular state. When the agent takes an action in a particular state it receives a reward (or penalty), and the goal of the agent is to maximise its total reward such that when taking an action it must also take the potential future reward into account. For Flappy Bird, the agent is the bird whose possible actions are to do nothing or flap, and whose current state is defined by

MIN/MAX ALGORITHM:

This is a classic game-playing algorithm that is commonly used for Tic Tac Toe. It is a recursive algorithm that is guaranteed to find the optimal move in Tic Tac Toe, but it can be computationally expensive as the game tree grows exponentially with the number of possible moves

ALPHA-BETA PRUNING:

This is an optimization of the minimax algorithm that reduces the number of game states explored by pruning branches of the game tree that are unlikely to lead to an optimal move. Alpha-beta pruning is a widely used technique to speed up game-playing algorithms like minimax, making them more efficient for real-time or resource-constrained environments

REINFORCEMENT LEARNING:

This is a ML approach where an agent learns to play Tic Tac Toe through trial-and-error interactions with the game environment. The agent receives feedback in the form of rewards or penalties based on its moves and aims to maximize the cumulative rewards over time.

PROPOSED SYSTEM

Our proposed system aims to develop a more efficient and reliable method for playing Tic Tac Toe optimally using a combination of techniques. The system will use the minimax algorithm, which is a well-known algorithm for decision making in two-player games, to determine the optimal move based on the current state of the gameIn addition to the minimax algorithm, we will use neural networks to improve the efficiency and accuracy of the decision-making process. The neural networks will be trained using a dataset of game states and their corresponding optimal moves to predict the best move in real-time.To further improve the system's performance, we will incorporate alpha-beta pruning, which is a technique for reducing the number of game states that need to be evaluated during the minimax algorithm. This will help reduce the computational cost of the system and enable faster decision-making.The proposed system will be evaluated against existing systems and tested on various scenarios to demonstrate its efficiency and reliability in playing Tik Tak Toe optimally. We believe that this system has the potential to be extended to other games and applications that require efficient and reliable decision-making.

4. TECHNIQUES & ALGORITHMS

REINFORCEMENT LEARNING:
One approach to developing a Tic Tac Toe game using ML is by using reinforcement learning. The algorithm learns to play the game by playing against itself or an opponent, and over time it learns which moves lead to a win and which lead to a loss. This involves a reward system, where the algorithm is rewarded for making good moves and penalised for making bad moves.

DECISION TREES:
Another approach to developing a Tic Tac Toe game is by using decision trees. The algorithm can use a decision tree to determine the best move to make based on the current state of the board. This approach involves creating a decision tree based on different board states, with each node in the tree representing a possible move.

NEURAL NETWORKS:
A third approach to developing a Tik Tak Toe game is by using neural networks. The algorithm can use a neural network to predict the best move to make based on the current state of the board. This involves training the neural network on a dataset of different board states and the corresponding best moves.

MONTE CARLO TREE SEARCH:
Another technique that can be used for developing a Tic Tac Toe game using ML is Monte Carlo Tree Search. This technique involves building a tree of possible moves and outcomes and then simulating many games to determine which move is most likely to lead to a win.

In summary, there are several techniques that can be used to create a Tic Tac Toe game based on ML, including reinforcement learning, decision trees, neural networks, and Monte Carlo Tree Search. The choice of technique depends on various factors such as the complexity of the game, the available resources, and the project goals.

5. IMPLEMENTATION

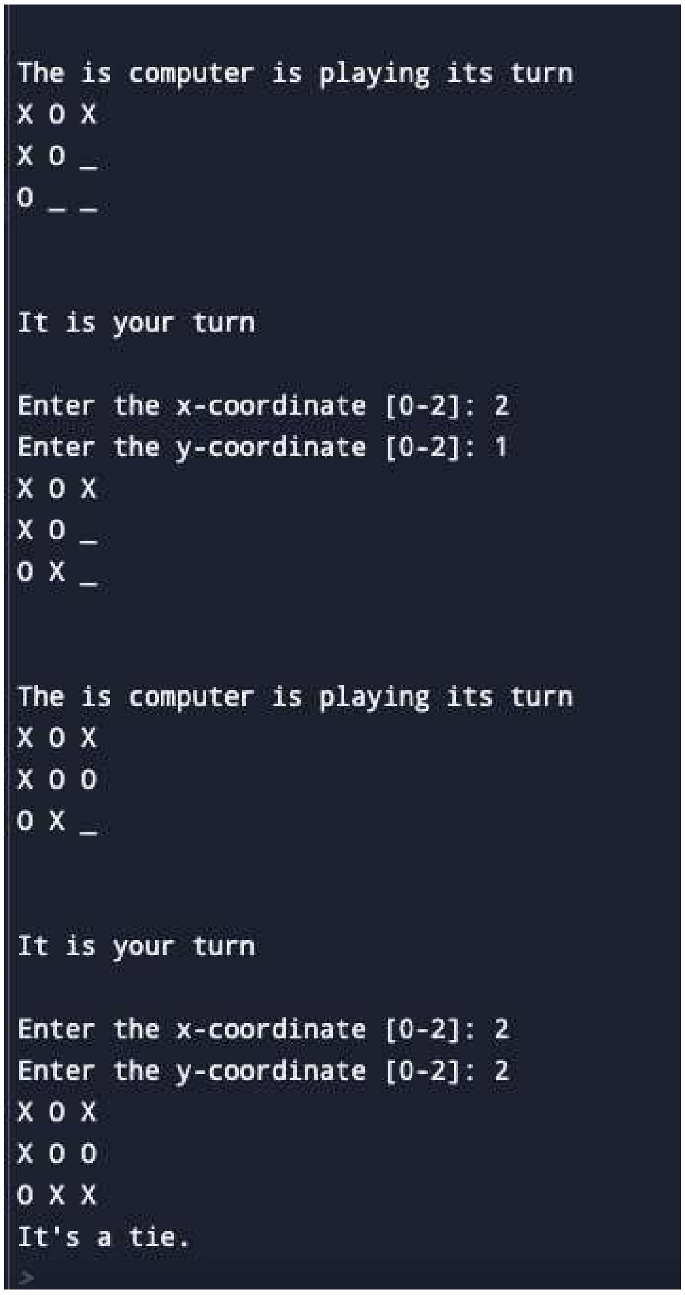
SOURCE CODE:

```
BOARD_EMPTY = 0
BOARD_PLAYER_X = 1
BOARD_PLAYER_O = -1 [
0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]

def print_board(s):   def
convert(num):     if num ==
BOARD_PLAYER_X:
      return 'X'     if num ==
BOARD_PLAYER_O:
      return 'O'     return '_'   i =
0   for _ in range(3):     for _
in range(3):
print(convert(s[i]), end=' ')
      i += 1
    print()
from collections import Counter

def player(s):
  counter = Counter(s)
x_places = counter[1]
o_places = counter[-1]

  if x_places + o_places == 9:
    return None   elif
x_places > o_places:
    return BOARD_PLAYER_O
else:
    return
BOARD_PLAYER_X def
actions(s):   play = player(s)
  actions_list = [(play, i) for i in range(len(s)) if s[i] ==
BOARD_EMPTY]   return actions_list def result(s, a):   (play, index) = a
s_copy = s.copy()   s_copy[index] = play   return s_copy def terminal(s):
for i in range(3):
    # Checking if a row is filled and equal.     if s[3 * i] == s[3 * i
+ 1] == s[3 * i + 2] != BOARD_EMPTY:       return s[3 * i]
    # Checking if a column is filled and equal.
    if s[i] == s[i + 3] == s[i + 6] != BOARD_EMPTY:
      return s[i]

  # Checking if a diagonal is filled and equal.
if s[0] == s[4] == s[8] != BOARD_EMPTY:
    return s[0]   if s[2] == s[4] == s[6] !=
BOARD_EMPTY:
    return s[2]

  # Checking if the game has no more moves available
if player(s) is None:     return 0

  # Return None if none of the previous conditions
satisfy.   return None def utility(s):   term = terminal(s)
  # Return who wins the game if the game has terminated
if term is not None:
    return term

  # Get the list of actions available
action_list = actions(s)   utils = []
for action in action_list:
    # Create a new state applying the action to current state
new_s = result(s, action)
    # Add the score of the new state to a list
utils.append(utility(new_s))

  score = utils[0]
play = player(s)
  # Calculate the max score if X is playing
if play == BOARD_PLAYER_X:     for i
in range(len(utils)):       if utils[i] > score:
score = utils[i]
  # Calculate the min score if O is playing
else:     for i in range(len(utils)):       if
utils[i] < score:         score = utils[i]
return score def utility(s, cost):   term =
terminal(s)   if term is not None:
    # Return the cost of reaching the terminal state
    return (term, cost)

  action_list = actions(s)
  utils = []   for action in
action_list:     new_s =
result(s, action)
    # Every recursion will be an increment in the cost (depth)
utils.append(utility(new_s, cost + 1))

  # Remember the associated cost with the score of the state.
score = utils[0][0]   idx_cost = utils[0][1]   play = player(s)
```

```
if play == BOARD_PLAYER_X:     for i in
range(len(utils)):     if utils[i][0] > score:        score =
utils[i][0]        idx_cost = utils[i][1]   else:
    for i in range(len(utils)):
if utils[i][0] < score:
score = utils[i][0]
idx_cost = utils[i][1]

  # Return the score with the associated cost.
return (score, idx_cost)  def minimax(s):
  action_list = actions(s)
utils = []   for action in
action_list:     new_s =
result(s, action)
    utils.append((action, utility(new_s, 1)))   # Each
item in utils contains the action associated   # the
score and "cost" of that action.

  # if utils has no objects, then return a default action and utility
if len(utils) == 0:     return ((0, 0), (0, 0))

  # Sort the list in ascending order of cost.
sorted_list = sorted(utils, key=lambda l : l[0][1])
  # Since the computer shall be Player O,   # It is safe
to return the object with minimum score.   action =
min(sorted_list, key = lambda l : l[1])   return action
if __name__ == '__main__':
  # Initializing the state
  s = [BOARD_EMPTY for _ in range(9)]   print('|-------
WELCOME TO TIC TAC TOE -----------|')   print('You
are X while the Computer is O')

  # Run the program while the game is not terminated
while terminal(s) is None:
    play = player(s)     if play ==
BOARD_PLAYER_X:
      # Take input from user       print('\n\nIt is
your turn', end='\n\n')       x = int(input('Enter the
x-coordinate [0-2]: '))       y = int(input('Enter the
y-coordinate [0-2]: '))       index = 3 * x + y

      if not s[index] == BOARD_EMPTY:
        print('That coordinate is already taken. Please try again.')
continue

      # Apply the action and print the board
      s = result(s, (BOARD_PLAYER_X,
index))       print_board(s)     else:
    print('\n\nThe is computer is playing its turn')       #
Get the action by running the minimax algorithm
action = minimax(s)
    # Apply the returned action to the state and print the board
    s = result(s, action[0])
print_board(s)

  # We know that terminal(s) is not None
  # determine the winner   winner =
terminal(s)   if winner ==
BOARD_PLAYER_X:     print("You
have won!")   elif winner ==
BOARD_PLAYER_O:
   print("You have
lost!")   else:     print("It's
a tie.")
```

7.

OUTPUT SCREENSHOT



```
The is computer is playing its turn
X O X
X O _
O _ _


It is your turn

Enter the x-coordinate [0-2]: 2
Enter the y-coordinate [0-2]: 1
X O X
X O _
O X _


The is computer is playing its turn
X O X
X O O
O X _


It is your turn

Enter the x-coordinate [0-2]: 2
Enter the y-coordinate [0-2]: 2
X O X
X O O
O X X
It's a tie.
>
```

8.

CONCLUSION

In conclusion, the implementation of the Tic Tac Toe game using ML algorithms is a challenging but exciting project that can help you to gain a deeper understanding of various ML techniques and their applications. By using ML algorithms, we were able to develop a computer player that can play the game optimally and provide a challenging opponent for human players.The project involves several key steps, such as data collection and preprocessing, model selection and training, integration with the game interface, and testing and evaluation.

The choice of the ML algorithm, programming language, and framework depends on several factors such as the complexity of the game and the project goals.The project can be further extended by implementing more advanced ML algorithms, developing a more sophisticated game interface, or adding new features to the game.

9. FUTURE WORK

- In the future, we can explore several directions to extend the project further, such as:
- Developing a multi-player version of the game using ML algorithms.
- Integrating the game with natural language processing (NLP) techniques to enable human players to interact with the computer player using natural language.
- Applying the same techniques to other games and applications, such as chess, checkers, or even selfdriving cars.